vQuery: A Platform for Connecting Configuration and Performance

Ilari Shafer
Carnegie Mellon University

Snorri Gylfason
VMware, Inc.

Gregory R. Ganger
Carnegie Mellon University

Abstract

Discovering the causes of performance problems in virtualized systems is often more difficult than without virtualization, because they can be caused by changes in infrastructure configuration rather than the user's application. vQuery is a system that collects, archives, and exposes configuration changes alongside fine-grained performance data, so the two can be correlated. It gathers configuration change data without modifying the systems it collects from and copes with platform-specific details within a general, graph-based model of Infrastructure-as-a-Service (IaaS) infrastructures. Configuration data collected from two VMware® vSphere™ environments reveals that configuration changes are frequent and involved, opening interesting new directions for configuration-aware performance diagnosis techniques.

1. Introduction

Consolidating computing activities onto shared infrastructures, such as in cloud computing and other virtualized data centers, offers substantial efficiency benefits for providers and consumers alike. But, it also introduces complexities when trying to understand the performance behavior of any given activity, since it can depend on many factors not present when using dedicated infrastructure. For example, the VMs used for the activity can migrate or be resized, or new VMs for other activities can be instantiated on shared hardware. As on-demand resource allocation (as in cloud computing) and automated configuration optimization (e.g., via VMware DRS[1]) grow more common, such factors increasingly create potentially confusing performance effects.

Traditionally, to understand application performance and diagnose performance problems, administrators and application engineers rely on resource usage instrumentation data from infrastructure runtime systems, such as time-sampled CPU utilization, memory allocated, and network packets sent/received. In VM-based infrastructures, the same data types can be captured for each VM as well. But, while such data exposes how resource usage changed at a given point in time, it offers little insight into why. Deducing why, so that one can decide what (if any) reactive steps to take, often is left entirely to the intuitions and experience of those involved in the diagnosis.

We believe an invaluable additional source of information should be captured and explicitly correlated with resource utilization data: the configuration history. Of course, any runtime infrastructure maintains its current configuration, and many log at least some configuration changes. Since configuration changes often cause performance changes, purposefully or otherwise, correlating the two should make it possible to highlight root causes of many problems automatically. In addition, the combination of the two offers the ability to expose powerful insights for system management and automation, such as which configuration changes usually improve performance and how particular problems were overcome in the past.

This paper describes our prototype system (called vQuery) for configuration change tracking and mining, together with initial experiences. vQuery collects time-evolving configuration state alongside fine-grained resource usage data from a VMware vCloud™-based infrastructure, stores it, and allows it to be queried. Configuration changes are captured by listening to vSphere's API and vCloud's internal update notifications. They are stored as a time-evolving graph of entities (e.g., VMs and physical hosts) as vertices and relationships as edges. This general approach avoids changes to the infrastructure software, accommodates a range of IaaS systems, and allows a range of configuration history queries.

We deployed vQuery on a local VMware software based private cloud (referred to as Carnegie Mellon's vCloud) as well as a VMware testbed, with positive initial results. We illustrate some of the power of configuration change history with interesting anecdotes and data from these deployments, and discuss challenges still ahead on this line of research.

The remainder of this paper is organized as follows. Section 2 explains what we mean by "configuration data" and configuration changes in more detail, including examples from VMware systems. Section 3 describes the design and current implementation of vQuery, focusing on how configuration data is captured, stored, and queried. Section 4 presents some data, early experiences, and anecdotes from vQuery deployments. Section 5 discusses our ongoing research on vQuery and exploiting configuration change data. Section 6 discusses related work.

2. Configuration Data and Changes

In a distributed computing infrastructure, various types of configuration are spread across files, databases, and within software. The word "configuration" often is used for concepts that include command-line flags to programs, OS-level settings, and the layout of virtual machines across computing resources. In this work, we focus on the last type: infrastructure-level properties that affect how virtualized environments, such as vSphere and vCloud, function and that reflect their current state.

Even this type of configuration is very heterogeneous. Some data are as simple as key-value pairs, but other data encodes lists, objects, and hierarchies. Some is controlled by end users (e.g., the choice of guest operating system for a VM), some is primarily automated by the computing infrastructure (e.g., which IP address a VM is assigned), and some can be managed by both (e.g., the choice of physical resources that back a VM). More concretely, Table 1 shows a selection of configuration properties in a VMware-based environment, ranging from simple descriptive properties to relationships with other entities.

ENTITY TYPE	CONFIGURATION PROPERTIES
VM	**vSphere**: *host system*, *networks*, *datastore*, name, annotation, memory, vCPUs, CPU allocation (reservation/limit/shares), memory allocation (reservation/limit/shares), virtual disk layout (chain length), power state, guest OS type, guest OS state, guest OS screen dimensions, guest NIC (IPs, network, state), guest disk (capacity, free space, path), IP address, VMware tools state + version **vCloud**: *vApp*, *networks*, name, vCPUs, memory, guest OS, status, storage
vApp	**vCloud**: *owner*, *vDC*, *networks*, status
User	**vCloud**: name, VM quota
Host	**vSphere**: *network*, CPU (frequency, number of cores and packages), memory size, power state
Network	**vSphere**: name **vCloud**: fence mode, parent, DNS (addresses, suffix), netmask, IP ranges
Datastore	**vSphere**: name, capacity, free space, type, url

Table 1: selected configuration properties in vSphere and vCloud. The properties that represent other entities are shown in italic.

Modeling configuration consistently is one focus of vQuery. In addition to the different types and meanings of configuration within vSphere and vCloud, different virtualized infrastructures expose different configuration properties. For example, where vSphere has a platform-independent datastore abstraction, the OpenStack infrastructure platform separates storage into block storage, local storage, and a separate VM image service. We would like to represent configuration in a sufficiently general way to model such different environments.

A key aspect of configuration on which we focus is that much of it changes over time. Some configuration properties may change very slowly (e.g., the amount of RAM on a physical host is seldom adjusted), while others are increasingly dynamic (e.g., the placement of a VM on a physical host is adapted by DRS). In tracking configuration as it relates to performance, we focus on recording changes in order to ask questions such as "was there a relevant configuration change around the same time as a given performance change?" and "what were all the configuration changes associated with a given VM?" Beyond diagnosis, maintaining a change history can also help us understand how and why systems evolve[2].

Additionally, infrastructure configuration is not a collection of unrelated facts. Configuration properties are associated with entities, whether physical (hosts and physical networks) or virtual (VMs and users), and these entities are meaningfully related. For example, VMs are placed on physical hosts, and users own vApps, which contain VMs. Examples of these "relational" properties are shown in italic in Table 1. We believe maintaining information about the relational structure of configuration—and how it changes—is important for diagnosis. It is intuitively important to be able to ask questions, such as "which VMs were on a given host when there was a performance problem?" Additionally, a variety of diagnosis approaches have taken advantage of the fact that the effects of changes often propagate through causal dependencies among the components of a distributed system[3 4 5], many of which are directed along these relations.

3. vQuery: Design

vQuery is designed to track fine-grained configuration data in a way that maintains the features described above. At a high level, the problems we need to solve are the same as those for performance monitoring: how to collect, store, and access configuration data. A simplified overview of our approach is shown in Figure 1, and this section describes each component in turn.

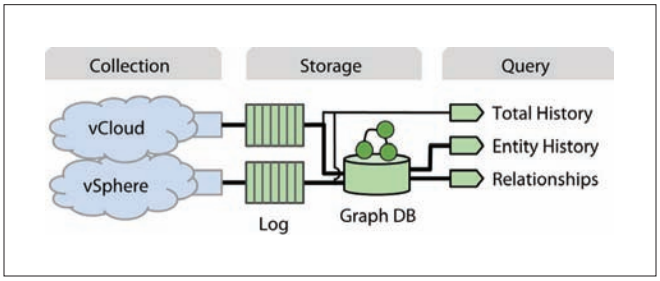

Figure 1: high-level overview of the vQuery configuration tracking system. It collects data from vSphere and vCloud, stores them in separate logs, and ingests them into a graph database to be queried.

3.1 Configuration Collection

Changes to configuration occur from both human and automated sources, and they clearly do not happen only at a fixed interval. It is insufficient for just the current configuration to be exposed by infrastructure APIs. For the collection process to be more efficient

and accurate than polling, there must be some way of obtaining updates. Ideally, the mechanism should require minimal, if any, modification of the infrastructure. We built our prototype without modifying code in vSphere or vCloud.

For vSphere, we build on the existing interface for subscribing to update notifications. Specifically, we use a PropertyCollector and its WaitForUpdates method to receive changes to a set of configuration properties of interest.

Although vCloud does not currently offer such an interface, we collect configuration from vCloud by listening to internal messages as a signal for when to query its configuration API. As an example, vCloud sends a message to start an action (e.g, start a VM, (1) in Figure 2), which results in a message sent to an AMQP message bus (1) and actions in vSphere (2). When the task is finished, a completion message is posted to the message bus. Our configuration collector listens to the same AMQP message bus (3), filters to listen to only task completion messages, and queries an appropriate API to find details about configuration change after a task completes (4).

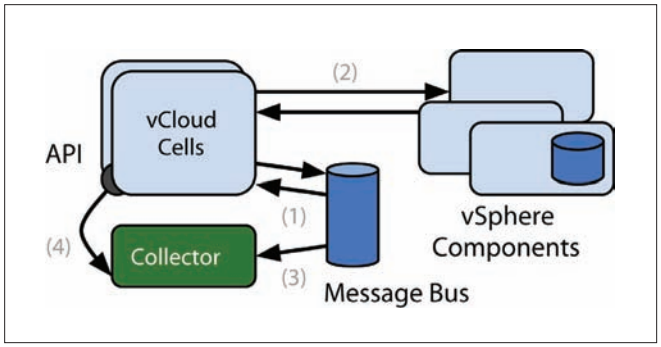

Figure 2: Configuration collection strategy for vCloud and OpenStack. The configuration collector listens to messages on the vCloud message bus and polls an appropriate API upon intercepting a task completion message.

This design pattern is not restricted to vCloud. The recently popular OpenStack IaaS also uses an AMQP message bus for inter-node communication[6]. We built enough of a collector for OpenStack to confirm that messages can be intercepted for configuration event notification. The same technique is not limited to message bus transports. For example, systems based on bare Remote Procedure Calls (RPC) could be instrumented similarly, albeit with a lower-level interceptor. In addition to vSphere, we have started collecting limited configuration data from an instance of the Tashi cluster management system[7].

In an ideal world, we would capture changes to all types of configuration that might affect performance, including those from the application layer. For example, recent research describes mechanisms for capturing changes to configuration files within guest VMs without modifying guest software[8]. Integrating such changes with those accessible from vCloud and vSphere is a direction for future work.

3.2 Configuration Storage

A primary challenge in storing configuration is how to represent it. Here, we describe a time-evolving representation of configuration information that is designed to support historical and relational queries using a general model. The representation is a graph with a loose schema—formally a typed, directed, attributed multigraph that also tracks time.

Infrastructure entities (VMs, physical hosts, storage nodes, networks, and so on) are vertices of this graph. Each vertex is associated with three mandatory fields: a unique identifier (id), a type (VM, host, and so on), and a valid-time interval[9]. Infrastructure entities can be created and removed over time (as in Figure 3, VMs can be allocated and deleted), but their historical presence must be remembered to support retrospective analytics. Each vertex also has a map of attribute names to a list of time-changing values ordered by time. The intuition behind this format is that each infrastructure attribute can change over time (e.g, a user changing the allocation of a VM, as shown as vRAM in Figure 3). The after-image of each value is appended to the list. Our implementation currently supports primitive types (strings, integers, floating-point numbers) and arrays thereof.

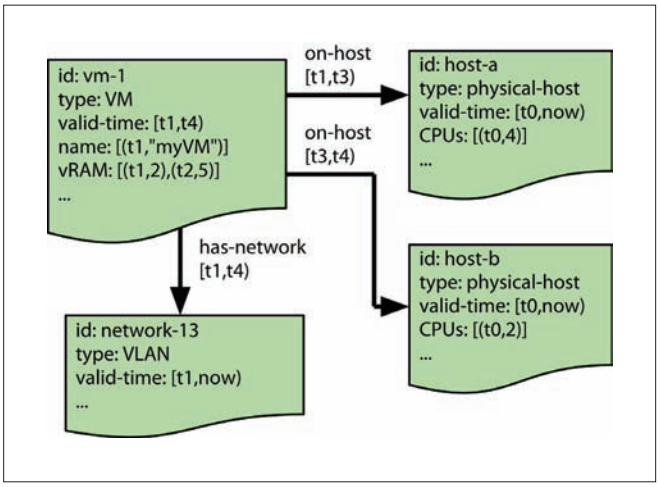

Figure 3: property graph of changing configuration. Each entity (VM, host, etc.) is associated with a number of time-evolving properties, in addition to time-evolving relationships with other entities. A unique identifier (id) and type of entity (type) are the only two required properties. The properties that track relationships (e.g. host) are specified by a user.

Edges between entities have two mandatory fields: a type of relationship and a valid-time interval. Similar to entities, such a relationship often exists only for a given time interval. For instance, in Figure 3, **vm-1** moved from **host-a** to **host-b** at time **t3** and was removed at time **t4**. In this way, the graph captures events such as VM migration not simply as events but as changes that relate entities. These edges—akin to foreign keys—are the schema of the configuration graph. Administrators must specify which configuration attributes have semantic meaning as dependencies (and must contain identifiers as values).

Maintaining configuration in this format also allows for the use of existing graph databases as an underlying persistence layer—in particular, those that store property graphs and have the ability to build indexes on properties.

Updating the graph, while relatively straightforward in principle, requires care in practice. The principle of the algorithm is reminiscent of that used to update a transaction-time state table in a temporal database[10], as applied to a property graph. Unfortunately, when receiving configuration updates from different layers of infrastructure, dependencies can be reported before the entities to which they refer. Consider the following ordered sequence of observations, similar to events observed in practice:

1. The VLAN **network-1** is created
2. vCloud reports that **vm-1** is connected to **network-1**
3. vSphere reports the existence of **network-1**

The final desired graph should contain a **has-network** edge from **vm-1** to **network-1**. If updates are applied in the given order, the graph will contain an invalid edge after step #2, since the existence of **network-1** is not yet known. We maintain a set of these "pending edges," which are scanned as new updates arrive. If one matches a newly-created entity the dependency is added with the original valid-time. As a beneficial side effect, this technique allows the update algorithm to operate with insertion batches atop the transactional graph database used (Neo4j[11]).

One drawback of storing configuration so generally is that we push the problem of forming meaningful queries to the querier. For example, retrieving a list of VMs requires selecting entities with the VM type rather than scanning a table named "VMs." Also, we assume loosely synchronized timestamps across different reporters of entity information, a property provided by the underlying VMware infrastructure.

3.3 Configuration Query

To ask questions about configuration history, we build a few abstractions on top of the graph database to supplement its query language[12]. Here, we focus on a few that align with our primary goals of historical queries that provide the history of an entity or the system, and relational queries that discover entities that likely depend on or influence each other.

- Historical: get-backlog(t_{start}, t_{end}): obtain all configuration changes to any entity between times t_{start} and t_{end}.
- Historical: get-property-names(*E*): get a list of properties associated with entity *E*, followed by get-property (*E*, *name*) to get a time-ordered list of changes to the property with name *n*.
- Relational: get-subgraph (*E*, *d*): do a breadth-first traversal of entities connected to entity *E*, up to a maximum depth *d* (or, with get-subgraph(*E*, *n*), up to a maximum number of entities *n*).

3.4 Performance Collection

In addition to the technique for storing configuration data described above, a source of performance data is necessary to connect configuration with performance. The performance data we consider consists of time series streams of metrics reported by the hypervisor and aggregated by management software. In contrast to configuration data, many mature systems exist for collecting and archiving this data at the infrastructure level[13 14 15].

For performance data collection, we use the StatsFeeder prototype described in more detail in the first issue of the VMware technical journal[16]. We collect nine metrics from each virtual machine and 15 metrics from each physical host every 20 seconds. These performance metrics are described briefly in Table 2.

VIRTUAL MACHINE	
CPU	**usage**: time used by this VM **system**: time spent in the VMkernel **wait**: time spent waiting for hardware/kernel locks **ready**: time spent waiting for a CPU (e.g., on an oversubscribed host) **guaranteed**: time used of the total guaranteed to the VM **extra**: time used beyond what the VM was originally assigned
Memory	**swapped**: amount of VM memory swapped out to disk **swaptarget**: amount of memory the VMkernel is aiming to swap **vmmemctl**: size of the memory balloon
HOST	
CPU	**usage**: aggregated time the CPU was used **idle**: time the CPU was idle
Disk	**usage**: average disk throughput **read**: average read throughput **write**: average write throughput **commands**: disk commands issued **commandsAborted**: disk commands aborted **busResets**: SCSI bus reset commands **numberRead**: number of disk reads **numberWrite**: number of disk writes
Network	**packetsRx**: packets received **packetsTx**: packets transmitted **usage**: average transmit + receive KB/s **received**: average receive KB/s **transmit**: average transmit KB/s

Table 2: collected performance data. All metrics are times, averages, or sums over a sample period (20s).

4. Early experiences with vQuery

A full evaluation of the vQuery framework would assess whether it can answer real diagnosis and monitoring queries. Although the project is still in the preliminary stage, this section provides some early experiences with configuration data collection and synthetic relational queries.

4.1 Historical

A functional configuration monitor collects and stores configuration changes over extended periods. This section describes some of the output from the two vSphere instances to which vQuery has been connected. The Virtual SE Lab (vSEL)[17] is an environment at VMware that is used for events at VMworld, training, and demos. We collected configuration changes from it a month prior to VMworld 2011. The vCloud at Carnegie Mellon (CMU) is a cloud we deployed to support academic workloads from courses, individual researchers, and groups with large research computing demands submitted via batch schedulers. Table 3 lists a few basic metrics of configuration change for each environment.

The last row of Table 3 highlights the diversity of configuration—and the need to be somewhat selective in what is collected and retained. One of the configuration properties exposed by vSphere and collected in the CMU dataset was datastore free space, a frequently updated property that accounted for over 63% of the configuration changes we observed. Although free space changes can be important to monitor, either collecting them infrequently or treating them as time series metrics (rather than as configuration changes) is more appropriate.

	vSEL	CMU
Collection period	12 days, starting 21 July 2011	75 days, starting 12 July 2012
Number of physical hosts	100	15
Number of changes	27888	63820
Number of configuration properties gathered	11	36
Number of changes, less free space changes	27888	23466

Table 3: Basic metrics of configuration change from two vSphere instances.

To better understand configuration changes that have occurred, visualization is crucial. As one example view, Figure 4 shows the configuration changes that occurred in the 75-day CMU dataset. Since there are so many types of configuration changes, we only show the top 10 types of change in the legend (by number of changes). A number of noteworthy events are visible from temporal and spatial groups on the chart. (See the caption for detail.)

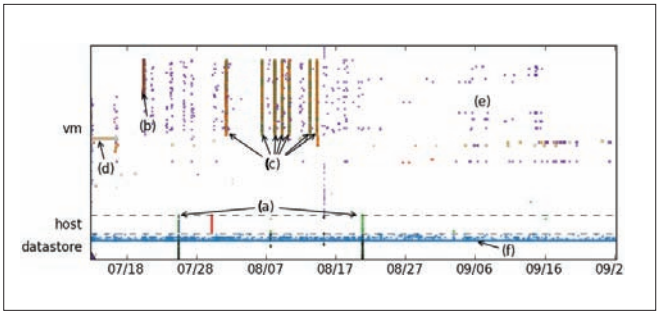

Figure 4: Configuration changes to a small datacenter. Dots represent configuration events to VMs, hosts, and datastores (spread on the vertical axis) across the horizontal time axis. The labeled periods are:

a) changes to many hosts and datastores around the time of a switch outage (first event) and switch replacement (second event) in another virtual datacenter

b) a user adding 30 VMs to an existing set of VMs to run experiments

c) the same user restarting the entire set of VMs when they became unresponsive

d) a user setting up a Windows VM, including many restarts

e) many points in this region (and between (b) and (c)) are VM migrations

f) this row of changes is primarily changes to datastore free space. (The VM disk free space changes shown in Table 3 are filtered out of this image.)

An additional observation we make about the snapshot of configuration in Figure 4 is that many configuration changes co-occur. For example, when VMs are restarted (e.g., the events marked as (c)), their power state changes along with the status of VMware tools in the guest OS and the status of its connection to a virtual network. Together, these changes represent the event "VM restart". Attributing its performance effects to a single one of these changes (particularly a change such as the state of VMware tools) would be misleading. Together with the observation in Table 3 that some configuration events are less meaningful than others, distilling semantically meaningful changes from the noise in configuration will be an important step forward.

4.2 Relational

One important aspect of vQuery is providing query access to related entities, which builds on database support for rapid neighborhood queries in the spatio-temporal configuration graph. We use a graph database (Neo4j); these databases are typically optimized for fast constant-time adjacency lookup[18]. This feature is one key way to manage queries across large graphs: the entities that are closer through dependency traversal are those that are more likely related.

For example, when performing a diagnosis query involving the performance of VM v, likely culprits include configuration changes to its resources (e.g., compute, networking, storage), which are within a traversal distance of 1. Furthermore, other VMs contending for those resources are also of interest, and are within a distance of 2. Although infrequent, relationships with a distance of 3 also arise: VMs in vCloud are modeled as abstract entities that are backed by VMs in vSphere.

Correlating configuration changes from a vCloud VM to a colocated vSphere VM needs 3 hops. If one needs to connect configuration changes to another vCloud VM, the distance would be 4 hops. Most cases involve just 1-2 hops.

To demonstrate that queries in common cases are relatively fast, Figure 5 shows the time required to run a query starting over the largest portion of the vSEL configuration graph. We run queries starting from a random entity in the 1821-entity graph up to a given depth. One can observe that querying for entities separated by a distance of 1 is fast (typically less than two milliseconds), and queries to distance 2 are typically under 10ms.

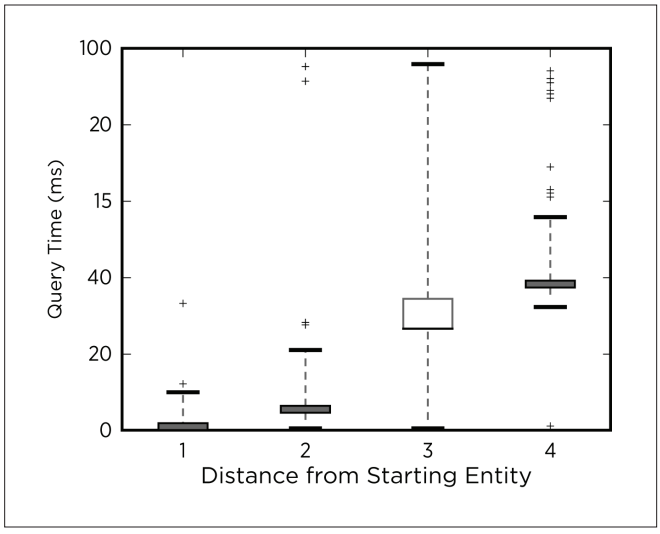

Figure 5: time taken to retrieve related entities to a random starting entity. 1,000 queries were performed at each distance, and boxes extend to the interquartile range.

5. Next Steps

As described above, vQuery forms an infrastructure for collecting, storing, and querying fine-grained configuration and performance data. Moving forward, we plan to use these augmented sources of monitoring data to perform more accurate diagnoses than with traditional black-box performance data alone. In particular, our next aim is to find configuration changes that are the root cause of performance problems. Three concrete examples are:

- **Short-term changes**. VM migration performed by DRS and virtual disk migration performed by storage DRS require network bandwidth and physical host resources. By monitoring performance, we hope to observe the short-term impact of these mechanisms and attribute it to the host and storage configuration changes we observe. We believe the fine-grained performance information we collect will be important to distinguish these performance variations, in addition to recent historical configuration
- **Contention**. virtualized workloads contend for resources, and perfect isolation is not yet a reality across resources, such as caches and disks. Migrating or starting workloads that use a host, datastore, or network can be a source of performance variation for VMs sharing that resource. The configuration changes we measure include migrations and power state changes, which we hope to correlate with performance monitoring data of contending entities. We believe relational queries will be necessary to identify configuration changes that occur to "neighbors," which are potential sources of contention.
- **Explaining parameters**. simply understanding which performance metrics are influenced by a configuration change can be a valuable source of guidance when identifying configuration-related problems, since the impact of configuration parameters often is unclear from name or documentation alone. Identifying performance changes related to configuration could allow us to annotate configuration parameters with the metrics they affect, providing guidance towards how they behave.

6. Related Work

6.1 Configuration Management

Recognition of the complexity of deploying and managing applications across clusters has spurred many configuration management efforts. Tools that have received recent attention include Chef[19] and Puppet[20], which focus on automated application deployment and configuration. CFengine[21] was among the first such tools, designed to reduce the burden of manually scripting policies and configurations across Unix workstations. It has since added support for deploying policies across the cloud computing environments we consider.

These tools primarily facilitate the creation of configuration rather than monitoring changes over time. That is, most focus on actuating configuration rather than monitoring what exists. CFengine is notable among the examples above for also incorporating a familiar-sounding notion of "knowledge management," which is a collection of facts about infrastructure and the relationships between them.

6.2 Correlating Configuration with Performance

Much work on understanding the connection between configuration and performance is focused on tuning configuration to optimize application performance. At least a few techniques, though, focus on our primary motivating use case: finding configuration changes that are the root cause of performance changes.

Many of these techniques have emerged from work on diagnosis in large-scale networks. MERCURY[22] considers an instance of the problem in ISP networks, and identifies the impact of upgrades and routing configuration changes on time series performance indicators, such as CPU utilization and packet loss. Whereas MERCURY considers mostly long-term changes in performance, PRISM[23] operates in the same setting and focuses instead on shorter time-scale changes, such as "spikes." WISE[24] also operates on ISP configuration and performance, but uses it to answer questions of the form "what would be the performance impact of making a configuration change?"

In the context of distributed applications, although NetMedic[25] uses two known snapshots in time as "good" and "problematic" points for diagnosing application-level errors, it uses some of the same concepts discussed here—notably, inference based on system performance data and an (automatically generated) dependency graph. ASDF[26] also correlates multiple time evolving measurements, similar to the black-box monitoring data described here, to perform root-cause diagnosis of performance problems.

6.3 Problem Diagnosis

Our work shares high-level goals with efforts to diagnose problems in distributed systems using widely available black-box performance metrics, such as CPU time and network throughput. For instance, Kasick et al. use statistical comparison across multiple machines to perform root-cause diagnosis in parallel file systems[27]. At the application level, work focused on multi-tier distributed systems has used time series CPU performance metrics to localize faults to individual machines[28], and domain-specific counters in IP networks[29].

By taking advantage of deeply instrumented "white-box" systems, a broader range of distributed system diagnosis techniques have been used for finding the sources of performance problems. For example, end-to-end traces , which track activity as it moves across system components, can be a rich source of insight[30]. Spectroscope[31] is one such tool that leverages these traces for root-cause performance problem diagnosis.

7. Summary

In virtualized environments, such as VMware vSphere, the additional indirection between workloads and the resources they use can lead to additional challenges when finding the source of performance problems. Infrastructure configuration changes can be a hidden source of performance variation. Identifying such effects requires configuration change capture and analysis. vQuery is a system for tracking configuration changes so that we can correlate them with traditional performance data, and early experiences with it are promising. Moving forward, we plan to integrate the data we collect to automatically produce insight about configuration-related performance problems in virtualized infrastructures.

References

1 A. Gulati, A. Holler, M. Ji, G. Shanmuganathan, C. Waldspurger, and X. Zhu. "VMware distributed resource management: design, implementation, and lessons learned." VMware Technical Journal, vol. 1, no. 1, pp. 47–64, 2012.

2 H. Kim, T. Benson, and A. Akella. "The Evolution of Network Configuration: A Tale of Two Campuses," IMC 2011.

3 A. A. Mahimkar, H. H. Song, Z. Ge, A. Shaikh, J. Wang, J. Yates, Y. Zhang, and J. Emmons, "Detecting the performance impact of upgrades in large operational networks." ACM SIGCOMM Computer Communication Review, vol. 40, no. 4, pp. 303–314, 2010.

4 P. Bahl, R. Chandra, and A. Greenberg, "Towards highly reliable enterprise network services via inference of multi-level dependencies," ACM SIGCOMM 2007.

5 M. Y. Chen, A. Accardi, E. Kiciman, J. Lloyd, D. Patterson, A. Fox, and E. Brewer, "Path-based failure and evolution management," NSDI 2004.

6 OpenStack, http://www.openstack.org.

7 M. A. Kozuch, M. P. Ryan, R. Gass, S. W. Schlosser, D. O'Hallaron, J. Cipar, E. Krevat, J. López, M. Stroucken, and G. R. Ganger, "Tashi: location-aware cluster management," ACDC 2009.

8 W. Richter, M. Satyanarayanan, J. Harkes, and B. Gilbert, "Near-Real-Time Inference of File-Level Mutations from Virtual Disk Writes," Technical Report CMU-CS-12-103, 2012.

9 C. S. Jensen, J. Clifford, S. K. Gadia, A. Segev, and R. T. Snodgrass, "A Glossary of Temporal Database Concepts," ACM SIGMOD Record, vol. 21, no. 3, pp. 35–43, Sep. 1992.

10 R. T. Snodgrass, Developing time-oriented database applications in SQL. Morgan Kaufmann Publishers Inc., 1999.

11 Neo4j, http://neo4j.org/

12 Neo4j cypher query language, http://docs.neo4j.org/chunked/stable/cypher-query-lang.html

13 Nagios, http://www.nagios.org

14 Zenoss, http://www.zenoss.com/

15 IBM Tivoli, http://www.ibm.com/developerworks/tivoli/

16 V. Soundararajan, B. Parimi, and J. Cook, "StatsFeeder: An Extensible Statistics Collection Framework for Virtualized Environments," VMware Technical Journal, vol. 1, no. 1, pp. 32–44, 2012.

17 F. Donald. "cim1436 - Virtual SE Lab (vSEL): Building the VMware Hybrid Cloud." VMworld 2011.

18 M. Rodriguez. "MySQL vs. Neo4j on a Large-Scale Graph Traversal," http://java.dzone.com/articles/mysql-vs-neo4j-large-scale

19 Opscode Chef, http://www.opscode.com/

20 Puppet, http://puppetlabs.com/

21 M. Burgess, "A site configuration engine," Computing systems, vol. 8, no. 2, pp. 309–337, 1995.

22 A. Mahimkar, H. H. Song, Z. Ge, A. Shaikh, and J. Wang, "Detecting the Performance Impact of Upgrades in Large Operational Networks," SIGCOMM 2010.

23 A. Mahimkar, Z. Ge, J. Wang, and J. Yates, "Rapid detection of maintenance induced changes in service performance," CoNEXT 2011.

24 M. Tariq, A. Zeitoun, V. Valancius, N. Feamster, and M. Ammar, "Answering what-if deployment and configuration questions with WISE," in ACM SIGCOMM Computer Communication Review, 2008, vol. 38, no. 4, pp. 99–110.

25 S. Kandula, R. Mahajan, P. Verkaik, S. Agarwal, J. Padhye, and P. Bahl, "Detailed diagnosis in enterprise networks," ACM SIGCOMM Computer Communication Review, vol. 39, no. 4, p. 243, Aug. 2009.

26 K. Bare, S. Kavulya, J. Tan, X. Pan, E. Marinelli, M. Kasick, R. Gandhi, and P. Narasimhan, "ASDF: An Automated, Online Framework for Diagnosing Performance Problems," in Architecting Dependable Systems VII, A. Casimiro, R. de Lemos, and C. Gacek, Eds. Springer Berlin / Heidelberg, 2010, pp. 201–226

27 M. Kasick, J. Tan, R. Gandhi, and P. Narasimhan, "Black-box problem diagnosis in parallel file systems," FAST 2010.

28 K. A. Bare, S. Kavulya, and P. Narasimhan, "Hardware performance counter-based problem diagnosis for e-commerce systems," NOMS 2010.

29 S. P. Kavulya, S. Daniels, K. Joshi, M. Hiltunen, R. Gandhi, and P. Narasimhan, "Draco : Statistical Diagnosis of Chronic Problems in Large Distributed Systems," DSN 2012.

30 E. Thereska, B. Salmon, J. Strunk, M. Wachs, M. Abd-El-Malek, J. Lopez, and G. R. Ganger, "Stardust: Tracking activity in a distributed storage system," SIGMETRICS 2006.

31 R. R. Sambasivan, A. X. Zheng, M. D. Rosa, E. Krevat, S. Whitman, M. Stroucken, W. Wang, L. Xu, and G. R. Ganger, "Diagnosing performance changes by comparing request flows," NSDI 2011.